**Tuteurs informatique**
**Nous contacter**

Utiliser ce site
Actu et stages
Docs à imprimer
Plan du site
Rechercher

Les tuteurs
Unix
Accents
Aide
Archiver
Chercher
Clavier, souris
Autres utilisateurs
Config conscrits
Copyrights
Disquettes
Divers
Droits d'accès
Éditeurs
Exercices
**Fichiers**
Mémo formats
Histoire
Imprimer
Jamais !
Jeux
Se loguer
Supports de données amovibles
Place disque
Unix
Processus
Répertoires
Shell
Systèmes de fichiers
Machines Unix

Serveur des élèves
ENS
SPI

# Manipuler les fichiers

- Créer un fichier
  - Généralités
  - Créer un fichier texte
  - Créer un fichier vide
- Nommer ses fichiers
- Lire un fichier
  - Lire un fichier avec `cat`
  - Les *pagers* `more` et `less`
  - Afficher le début d'un fichier
  - Afficher la fin d'un fichier
- Manipuler les fichiers
  - Copier un fichier
  - Renommer un fichier
  - Déplacer un fichier
  - Effacer un fichier
  - Concaténer des fichiers
- Récapitulons (exemple)

## Créer un fichier

### Généralités

Un fichier est un *paquet d'informations* : ce peut être aussi bien du texte, que de la musique ou une image. Chaque objet que vous manipulez en informatique (une page Web, la musique de Casimir, vos photos de vacances, un logiciel, etc.) correspond à un fichier.

Quand vous écrivez un texte avec OpenOffice.org et que vous le sauvegardez sous le nom « Rapport de maîtrise », vous créez un fichier à ce nom, écrit sur le disque dur. Chaque fois que vous écrivez quelque chose sur le disque, vous créez un ou plusieurs fichiers.

### Créer un fichier texte

Une des opérations les plus courantes consiste à écrire du texte. Pour cela, voir le cours sur les éditeurs de texte. Quand on vous demande de créer un fichier, généralement il faut utiliser un éditeur de texte. Ainsi, pour créer le fichier `index.html` (accueil d'une page Web), utilisez l'éditeur de KDE, kwrite, et tapez :

```
brick ~/www/ $ kwrite index.html
```

Si vous êtes pressé, la commande `cat` peut faire office d'éditeur minimaliste. Tapez la commande suivante, et quelques mots :

```
brick ~ $ cat > notes
Faire les courses
Passer à la bibliothèque
```

Et finissez par `Ctrl+D`. Le fichier `notes` contient les quelques lignes que vous avez saisies.

### Créer un fichier vide

On peut aussi créer un fichier à l'aide de la commande `touch`. `touch toto` crée un fichier vide appelé « toto ». En fait, cette commande sert essentiellement à modifier la date de dernière modification d'un fichier.

**Application :** vous avez une page Web sur le serveur des élèves et que vous ne voulez pas qu'à chaque virgule modifiée, on voie votre page dans la liste des pages récemment modifiées. Créez alors dans votre répertoire `www` un fichier vide appelé `.last_modified` :

```
brick ~/www/ $ touch .last_modified
```

Votre page, désormais, ne figurera dans la liste des récentes modifications qu'après un nouveau `touch` du fichier `.last_modified`.

## Nommer ses fichiers

Les noms de fichiers s'écrivent avec des caractères alphanumériques (lettres et chiffres). On peut utiliser des majuscules et des minuscules, mais elles ne sont pas interchangeables : `document` et `Document` sont deux fichiers distincts.

Il est préférable d'éviter les espaces dans les noms de fichiers. On peut les remplacer par un *underscore* (_) ou un tiret (-). :

```
banque-de-france.tex
banque_centrale.dvi
```

Il vaut mieux éviter les caractères accentués, et proscrire le reste (ponctuations diverses, signes bizarres). Il faut aussi éviter de faire commencer un fichier par un . (par exemple `.document`). Il est également sage d'éviter de

donner des noms trop longs (même si en théorie vous avez droit à 255 caractères).

Certains fichiers ont des suffixes conventionnels qui les identifient auprès des utilisateurs et de certains programmes (liste non exhaustive de suffixes et d'outils) :

| Suffixe | Nature | Logiciel à utiliser |
|---|---|---|
| `.tex` | Fichier source pour LaTeX | Éditeur de texte |
| `.dvi` | Résultat de la compilation LaTeX | `kdvi`, `xdvi` |
| `.aux`, `.log`, `.toc`, `.mp`, etc. | Fichiers produits automatiquement par la compilation LaTeX | À regarder avec `less` |
| `.ps`, `.eps` | Fichiers PostScript | `kghostview`, `gv` |
| `.txt` | Fichier texte | Éditeur si vous voulez le modifier, `less` si vous voulez juste le lire. |
| `.html`, `.shtml`, `.htm` | Pages Web | Navigateur (`konqueror`, `firefox`, `w3m`...) |
| `.au`, `.ogg`, `.mp2`, `.mp3` | Son | Les fichiers `.au` s'écoutent avec `play`, les `.ogg` avec `ogg123`, les `.mp3` avec `mpg123`. |
| `.tiff`, `.png`, `.jpg`, `.gif`, `.ppm`... | Images | `kview` pour les regarder ; `gimp` pour les modifier |
| `.mov`, `.avi` | Séquences animées | `mplayer` |
| `.pdf` | Portable Document Format | `kpdf`, `acroread` (Acrobat Reader) |
| `.gz`, `.tgz`, `.tar`, `.tar.gz`, `.Z`, `.tar.Z`, `.zip`, `.zoo`, `.lha`, `.lzx`, `.z`, `.uu`, `.shar` | Divers formats d'archives plus ou moins compressés | Consultez la page sur la compression |

# Lire un fichier

## Lire un fichier avec `cat`

Pour pouvoir consulter un fichier (sans l'éditer), le plus simple est la commande `cat` :

```
brick ~ $ cat fichier
```

Le problème est que `cat` affiche l'intégralité du fichier que vous lui demandez. Si le fichier est plus long que la taille de votre écran (ce qui arrive quand même souvent), vous verrez défiler très vite à l'écran le contenu de votre fichier, sans pouvoir l'arrêter ou le ralentir. Pas très pratique...

## Les *pagers* `more` et `less`

Mieux vaut alors utiliser ce qu'en anglais on appelle un *pager* (parfois traduit par « visionneuse » en français). Le nom « pager » vient du fait qu'il affiche un fichier page par page et non tout d'un coup comme le fait `cat`. Sur les Unix un minimum récents, la commande qui fait cela est `less`. Pour lire le fichier `toto.txt`, tapez simplement :

```
brick ~ $ less toto.txt
```

Voici un tableau des commandes utiles de `less` :

| | |
|---|---|
| `h` | afficher une aide. |
| `SPC` | faire défiler le texte page par page |
| `d`, `^D` (*down*) | descendre d'un demi-écran |
| `f`, `^F` (*forward*) | descendre d'un écran |
| `p`, `^P` (*previous*) | remonter d'un demi-écran |
| `b`, `^B` (*back*) | remonter d'un écran |
| `j` | descendre d'une ligne |
| `k` | remonter d'une ligne |
| `G` | aller à la fin du texte |
| *nombre*`G` | aller à la ligne de numéro *nombre* (par exemple, `1G` place à la première ligne du fichier). |
| `/mot` | chercher le « mot » dans le texte. Il sera signalé en inverse dans tout le texte. Pour aller à la prochaine occurence du mot, taper `n`, pour aller à la précédente occurence, taper `N`. |
| `q` | quitter |

**`more`** (en savoir *davantage* sur un fichier) est une commande standard sur tous les systèmes Unix (alors que

`less` n'est pas installé partout). `less` (installé à l'École) est cependant plus performant : les mots recherchés apparaissent sur fond noir au lieu d'être juste placés en haut de l'écran, le défilement du texte est plus précis. Si vous n'avez que `more` à disposition, vous pouvez utiliser les commandes suivantes :

| | |
|---|---|
| `h` | afficher une aide |
| `d` ( *down*) | descendre d'un demi-écran |
| `f` ( *forward*) | descendre d'un écran |
| `b` ( *back*) | remonter d'un demi-écran |
| `/mot` | chercher le « mot » dans le texte (le résultat est beaucoup moins clair qu'avec `less` ). |
| `q` | quitter |

## Afficher le début d'un fichier

On utilise la commande `head` pour afficher le début d'un fichier :

```
brick ~ $ head point_de_lendemain.txt
J'aimais éperdumment la comtesse de ... ; j'avais vingt ans, et
j'étais ingénu ; elle me trompa, je me fâchai, elle me
quitta. J'étais ingénu, je la regrettai ; j'avais vingt ans, elle
me pardonna : et comme j'avais vingt ans, que j'étais ingénu,
toujours trompé, mais plus quitté, je me croyais l'amant le mieux
(...)
```

Par défaut, `head` affiche les 10 premières lignes. Pour lui spécifier un nombre de lignes précis, utilisez l'option `-n`, *n* étant le nombre de votre choix :

```
brick ~ $ head -2 point_de_lendemain.txt
J'aimais éperdumment la comtesse de ... ; j'avais vingt ans, et
j'étais ingénu ; elle me trompa, je me fâchai, elle me
brick ~ $
```

« Mais à quoi est-ce que ça peut bien servir ? », vous direz-vous. Là, comme ça, c'est vrai que l'utilité est limitée. Mais en combinaison avec d'autres commandes, c'est très utile. Un exemple tiré de notre page sur la place disque :

```
sas ~ $ du -ak | sort -nr | head -10
98280   .
22007   ./www
1658    ./Mail
9637    ./www/hist
5005    ./www/hist/maitrise
4910    ./histoire
3941    ./Mail/archives
3757    ./www/philo/
3274    ./www/philo/cours
3112    ./www/philo/cours/hegel
2841    ./prive
```

Nous avons ici les 10 fichiers ou répertoires les plus gros de notre compte. Toutes les explications sur cette suite de commandes se trouvent dans nitre page sur la place disque.

## Afficher la fin d'un fichier

Cette fois, c'est la commande `tail` qu'on utilise, avec les mêmes options que `head` :

```
brick ~ $ tail -3 point_de_lendemain.txt
sensible. Adieu, encore une fois. Vous êtes charmant... Ne me
brouillez pas avec la Comtesse ». Elle me serra la main, et
me quitta.
```

Cette commande devient très intéressante avec l'option `-f` : celle-ci dit à `tail` de ne pas s'arrêter à la fin du fichier, mais de continuer à afficher les nouvelles informations qui seraient ajoutées au fichier. Une applications typique est la surveillance des fichiers de logs (journaux où le système, ou certains programmes, consignent des informations sur leur travail).

### Application

Vous êtes en train de rédiger votre `.htaccess` (fichier de configuration du serveur Web, Apache) et vous voulez regarder en temps réel les messages d'erreur du serveur.

Ces messages d'erreur, à l'ENS, sont dans `~webmaster/logs/access_log.curr`, il faut donc surveiller les messages d'erreur sui arrivent au fur et à mesure que vous faites vos essais. Tapez dans l'une de vos fenêtres la commande suivante :

```
sas ~ $ tail -f ~webmaster/logs/access_log.curr | grep monlogin
```

et vous verrez s'afficher, au fur et à mesure et en temps réel, toutes les erreurs qui contiennent votre login, donc vos pages. Pour arrêter `tail` quand vous en avez assez, tapez `Ctrl+C`.

# Manipuler les fichiers

## Copier un fichier

Copier un fichier, c'est prendre un fichier déjà existant et en créer une copie rigoureusement identique sous un nom différent. On utilise pour cela la commande `cp` (comme *cp*). Pour copier `toto.txt` en `toto2.txt` :

```
brick ~ $ cp toto.txt toto2.txt
```

Notez qu'on ne peut pas se servir de `cp` pour copier un répertoire. Ainsi, si vous essayez de copier votre répertoire `www`, `cp` va se plaindre :

```
brick ~ $ cp www www2
cp: www: is a directory
```

**S'exercer :** pour vérifier que vous avez bien compris, faites les exercices sur `cp`.

## Renommer un fichier

Renommer un fichier, c'est lui donner un nouveau nom, sans rien modifier d'autre. On utilise pour cela la commande `mv` (comme *move*). Pour renommer `dissert.tex` en `dissertation.tex` :

```
brick ~ $ mv dissert.tex dissertation.tex
```

La même commande sert également, avec la même syntaxe, pour les répertoires. Ainsi, si vous voulez renommer votre répertoire `boulot` en répertoire `magistere` :

```
brick ~ $ mv boulot magistere
```

## Déplacer un fichier

C'est toujours la commande `mv` que l'on utilise. La syntaxe est la suivante : `mv fichier nouvel_endroit`. Ainsi, vous voulez déplacez le fichier `cv.html`, situé à la racine de votre compte, dans votre répertoire `www` :

```
brick ~ $ mv cv.html www
```

Autre exemple :

```
brick ~ $ mv fichier ..
```

déplace un fichier dans le répertoire père — si vous ne comprenez pas ce terme, reportez-vous au tableau qui illustre la commande `cd` dans la page sur les répertoires.

**S'exercer :** pour vérifier que vous avez bien compris, faites les exercices sur `mv`.

## Effacer un fichier

Sous Unix, effacer un fichier, c'est le détruire de façon **irréversible**. Pas de concept de poubelle dans laquelle on peut fouiller, comme sous Windows ou Mac OS. C'est la commande `rm` (comme *remove*) qui efface un fichier :

```
brick ~ $ rm fichier
```

Dans les configs conscrits, depuis un certain temps, `rm` vous demande de confirmer l'ordre d'effacement :

```
brick ~ $ rm fichier
rm: remove fichier (yes/no)?
```

En effet, le `rm` que vous utilisez fait en fait `rm -i`, comme vous le montrera la commande `which` :

```
brick ~ $ which rm
rm: aliased to nocorrect rm -i
```

Cette précaution évite le fameux doigt qui glisse et autres erreurs qui effacent par mégarde des fichiers importants. Mais cette précaution n'existe pas pour toutes les configurations : observez donc la plus grande prudence en utilisant `rm`.

**Remarque** : à l'ENS, les partitions utilisateurs sont sauvegardées toutes les nuits par le SPI. Si donc vous effacez un fichier accidentellement et qu'il avait plus d'un ou deux jours, vous pouvez envoyer un courrier électronique au SPI pour demander qu'on vous le récupère.

Il existe aussi RCS, qui enregistre des copies de sauvegarde de vos fichiers, et permet de gérer les différentes versions de votre travail.

### Effacer plusieurs fichiers en même temps

Nettoyage de printemps sur votre compte, vous voulez supprimer plusieurs fichiers en même temps. Il suffit tout simplement de les indiquer à la suite à `rm` :

```
brick ~ $ rm article_vn.tex article_leveque.tex article_leveque2.tex
```

### Effacer un répertoire et son contenu

La commande pour effacer un répertoire est `rmdir` (cf. notre page sur les répertoires), mais elle ne marche que si le répertoire est vide. La commande `rm` ne peut pas effacer directement un répertoire :

```
brick ~ $ rm www
rm: www is a directory
```

Pour effacer un répertoire *et son contenu*, utilisez l'option `-R` (comme *recursive* de `rm`. Pour effacer votre répertoire `toto/` et les fichiers qu'il contient, tapez :

```
brick ~ $ rm -R toto
```

À l'ENS, parce que `rm` est en fait `rm -i`, `rm` vous demandera confirmation pour chacun des fichiers contenus dans le répertoire `toto/` :

```
brick ~ $ rm -R toto
rm: examine files in directory toto (yes/no)? yes
rm: remove toto/titi (yes/no)? y
rm: remove toto/tata (yes/no)?
(...)
```

Si vous trouvez fastidieux de devoir répondre oui à chaque fois, vous pouvez utiliser en plus l'option `-f` (comme *force*) de `rm`. Cette option supplante l'option `-i` et **efface irrémédiablement sans demander confirmation**. Faites donc très attention avec.

### Effacer des fichiers au nom exotique

Comment effacer des fichiers qui comment par des caractères bizarres, ou qui comportent des espaces dans leur nom ? La commande `rm` semble devenir subitement inefficace ! Essayons d'abord avec un fichier avec des espaces :

```
brick ~ $ rm siècle de périclès.doc
rm: siècle: No such file or directory
rm: de: No such file or directory
rm: périclès.doc: No such file or directory
```

En fait, `rm` croit que vous voulez effacer d'abord un fichier appelé `siècle`, puis un autre appelé `de`, etc. La solution est de mettre le nom du fichier entre guillemets ("), et `rm` l'efface sans problème :

```
brick ~ $ rm "siècle de périclès.doc"
remove siècle de périclès.doc? y
brick ~ $
```

Essayons maintenant avec un fichier commençant par un caractère bizarre, par exemple un tiret :

```
brick ~ $ rm -bla
rm: illegal option -- b
usage: rm [-f | -i] [-dPRrvW] file ...
       unlink file
brick ~ $ rm "-bla"
rm: illegal option -- b
usage: rm [-f | -i] [-dPRrvW] file ...
       unlink file
```

Pas bien brillant... La commande `rm` croit à cause du tiret initial qu'il s'agit d'options. Or il n'y a pas d'option `-b` pour `rm`. Solution : utiliser l'option `--` :

```
brick ~ $ rm -- -bla
remove -bla?
```

Une autre solution consiste à donner à `rm` un chemin, absolu ou relatif. Avec un chemin relatif, par exemple :

```
brick ~ $ rm ./-bla
remove -bla?
```

## Concaténer des fichiers

Concaténer des fichiers, c'est les placer bout à bout. On utilise la commande `cat` (comme *concatenate*) :

```
brick ~ $ cat fichier1 fichier2 > fichier3
```

Les fichiers 1 et 2 placés bout à bout et le résultat est mis dans un `fichier3` créé automatiquement. Le *shell* refuse d'écraser un fichier existant.

Dans `fichier3` apparaît d'abord `fichier1` puis `fichier2`. Concaténer deux fichiers de la sorte ne les détruit pas (il existe à la fois `fichier1`, `fichier2` et `fichier3`).

On peut aussi placer deux fichiers bout à bout sans créer de troisième fichier. La syntaxe est alors la suivante :

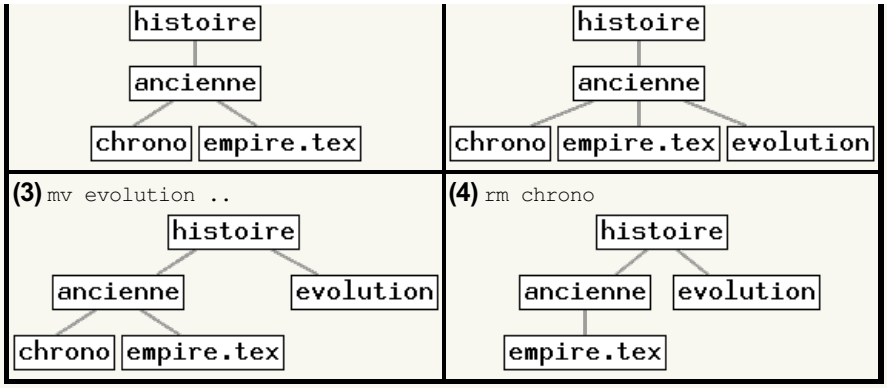
```
brick ~ $ cat fichier2 >> fichier1
```

`fichier2` est placé « au bout » de `fichier1`. `fichier2` reste inchangé, `fichier1` contient désormais son propre texte suivi du texte de `fichier2`.

# Récapitulons (exemple)

On veut renommer le fichier `chrono` du sous-répertoire `ancienne/` en `evolution`, et le mettre dans le répertoire `histoire/`. Dans cet exemple, nous sommes dans `histoire/ancienne/` :

| | |
|---|---|
| **(1)** État de départ : | **(2)** `cp chrono evolution` |




On peut aussi faire plus rapidement :

```
mv chrono evolution
mv evolution ..
```

Ou, en une seule ligne de commande :

```
mv chrono ../evolution
```

La même série d'opérations est possible depuis `histoire/`; elle se fera avec la suite de commandes suivantes :

| | |
|---|---|
| `cp ancienne/chrono evolution`<br>`rm ancienne/chrono` | On recopie le fichier `chrono` du sous-répertoire `ancienne/` en un second fichier `evolution` placé dans le répertoire courant : on se retrouve directement dans l'état 3. Ensuite, on efface le fichier `chrono` qui est toujours dans le répertoire `ancienne/`. |
| `mv ancienne/chrono ..`<br>`mv chrono evolution` | D'abord on déplace le fichier, ensuite on le renomme. |
| `mv ancienne/chrono evolution` | En une seule ligne, on déplace le fichier et on le renomme. |

Il y a donc rarement une seule façon de manipuler ses fichiers, c'est une question d'aisance et de rapidité.

Auteurs : Émilia Robin (2002), Marie-Lan Nguyen (2004). Mise à jour : Marc Mezzarobba. Dernière modification : 2013-10-20 par Antoine Amarilli.